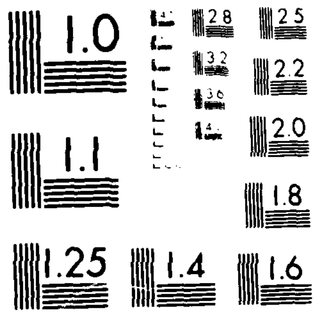
AD-A142 656 A NOTE ON THE ACCURACY OF TWO MICROPROCESSORS(U) 1/1
NATIONAL PHYSICAL LAB TEDDINGTON (ENGLAND) DIV OF
INFORMATION TECHNOLOGY AND COMPUTING B A WICHMANN
UNCLASSIFIED FEB 83 NPL/DITC-18/83 F/G 12/1 NL

NPL

END
DATE
FILMED
8-84
DTIC



MICROCOPY RESOLUTION TEST CHART

NPL

National Physical Laboratory

ISSN 0262-5369

National Physical Laboratory
Teddington, Middlesex TW11 OLW, UK

Approved on behalf of Director, NPL, by Mr E L Albasiny,
Superintendent, Division of Information Technology and Computing

NPL Report DITC 18/83
February 1983

A note on the accuracy of
two microprocessors

by
B A Wichmann

NPL Report DITC 18/83

February 1983

N A T I O N A L   P H Y S I C A L   L A B O R A T O R Y

A note on the accuracy of two microprocessors

by

B A Wichmann

Division of Information Technology and Computing

Abstract

An analysis has been performed of the accuracy of the mathematical functions in the Basic monitor of the ZX81 and BBC microprocessors. The tests used for this are those that are part of the NPL/University of Tasmania Pascal validation suite. The results of these tests shows that even the smallest of microprocessors can be expected to give mathematically sound values for the standard functions.

CONTENTS

Introduction

The Pascal validation suite contains tests for the accuracy of the mathematical functions derived from those of Cody and Waite for FORTRAN [1]. The Pascal Standard [2] requires that these functions deliver a result which is "an approximation to the corresponding mathematical result". (The Standard gives no other clue on the nature of the approximation; which could hence be unacceptable for certain applications.) The language standards for FORTRAN and Basic [3,4] do not require similar accuracy for these functions (although typically the same library is involved for a particular machine).

Changes had to be made from the FORTRAN version for the Pascal validation suite, for instance, to remove the error tests from the accuracy tests [5]. This particular change was necessary because a program containing an error such as evaluating sqrt(-1) can be rejected by a Pascal compiler before execution. Also, criteria were added to each test giving a pass/fail indication. The criteria used were based upon the results listed by Cody and Waite, being such that success was achievable with a small margin.

The purpose of running these tests in Basic was to see if the criteria for pass/fail were reasonable for the smallest of the microprocessor systems. The entire Basic system on the ZX81 resides in an 8K ROM, the floating point arithmetic operations being provided by software. The floating point package and the mathematical library is believed to be contained in no more than 2100 bytes [6]. Such testing cannot be undertaken on a regular basis, especially since the non-standard nature of most Basic systems results in a significant conversion effort.

The results of the tests are included in each section below. Most of the tests were successful and the cause of two of the errors in the unsuccessful tests on the ZX81 was located. Although the results on the BBC microprocessor were not as good, the author of the package, Dr A C Norman, has since modified the software to achieve essentially perfect results. Hence the conclusion is that the Pascal tests are realistic, even for the smallest of microprocessor systems.

Only the simplest of tests are included for the floating point routines themselves (ie the operations +, -, * and /). Very thorough tests are available [7] which have located errors in both hardware and software floating point units. However, the size of these tests (12000 lines) makes them inappropriate for Basic (and even Pascal).

The characteristics of the floating point system

The accuracies of the mathematical functions (such as square root) are judged in relation to the accuracy of the underlying floating point representation used. The Pascal validation suite test programs each include a separate copy of a routine MACHAR to determine a small set of fundamental constants termed "characteristics". (The separate copy is needed because the Standard does not adequately support separate compilation of procedures.) The procedure MACHAR is quite complex and typically accounts for half of the length of code of each test program.

The technique used for testing the two microprocessors was to run MACHAR once to determine the characteristics, which are then inserted directly into the other tests as required. This technique would not be appropriate for testing a variety of hardware without program modification (the situation for Pascal). Perhaps surprisingly, the two floating point systems are virtually identical. Hence the results of running MACHAR are presented together, explaining the differences as needed.

CHARACTERISTIC	DEFINITION and REMARKS
IBETA=2	The radix. A radix of 2 is known to be superior with respect to accuracy to the higher values used on some computers. The new IEEE standard [8] uses a binary radix.
IT=32	The number of binary digits in the floating point significand.
IRND=1	Floating point (addition) rounds. This characteristic has a significant impact on the rounding errors incurred.

NGRD=0 The number of guard digits for multiplication.

MACHEP=-32 The most negative integer such that the computed value of $1 + 2^{MACHEP}$ is different from unity

IEXP=8 The number of bits for the exponent (deduced from MINEXP).

MINEXP=-128 The most negative integer such that 2^{MINEXP} is a positive floating point number. This value implies that there is no gradual underflow.

MAXEXP=127 The largest integer such that 2^{MAXEXP} is a floating point number. This value is deduced from that of MINEXP, to avoid overflow.

$EPS=2^{-32}$ (about 2.328E-10)

The smallest integral power of two such that the computed value of 1+EPS differs from unity

$EPSNEG=2^{-31}$ (about 4.656E-10, ZX81);
$=2^{-32}$ (about 2.32E-10, BBC)

The smallest integral power of two such that the computed value of 1-EPSNEG differs from unity

$XMIN=2^{-128}$ (about 2.938E-39)

The smallest integral power of two that does not underflow.

$XMAX = 2^{127}$ (about 1.7014E38)

The largest finite representable floating point value.

The physical representation of floating point values takes five bytes. The first byte is the biased exponent. The first bit of the second byte is the sign and the remaining 31 bits are the bottom 31 bits of the mantissa. The most significant bit of the significand (S) is not stored.

Zero is stored as zero in all five bytes. If S is regarded as a fraction such that

$$^1/2 <= S < 1,$$

then the magnitude of the value is

$$S * 2^{e-128},$$

where e is the unsigned value of the first byte. The only valid value having the first byte zero is 0.0.

The Pascal version of MACHAR contains a dummy function "st" to force the storing of intermediate results. Inclusion of "st" avoids the otherwise incorrect results that would be obtained in the case of a computer with an "overlength accumulator". The problm here is that if the accumulator has superior properties to values stored elsewhere, then one must ensure that the properties of the stored values are observed. The superior properties of such an accumulator cannot be relied upon since in a high level language one does not have control over the accumulator. (This happened with the previous version of MACHAR when it was run on the Multics system.) Neither of the systems considered here has an overlength accumulator - its presence would require an increase in the size of a software floating point package of the Basic implementations.

An examination of the physical storage of values is possible by means of PEEK instructions. The examination is facilitated by placing the value to be inspected as the first variable (Z) whose address is then available in a fixed place within the monitor. Then to experiment with values, it is convenient to have the powers of 2 available in an array since the operator ** will not necessarily deliver the powers exactly. The value being considered can be read as a string and converted by the VAL operation so that the text of the expression can be displayed as well as the resulting bit pattern.

The program for the ZX81 is as follows:

PROGRAM TO DISPLAY VALUE IN BINARY ON THE ZX81

```
 10 LET Z=0                 - Z is the value to be examined
 20 DIM P(255)              - powers of 2, P(129+I)=2**I
 30 LET X=1                 - to calculate the negative powers of 2
 40 LET Y=1                 - to calculate the positive powers of 2
 50 LET P(129)=1            - = 2 ** 0
 60 FOR I=1 TO 126
 70 LET X=X/2
 80 LET Y=Y*2
 90 LET P(129+I)=Y
100 LET P(129-I)=X
110 NEXT I
120 LET P(2)=P(3)/2
130 LET P(1)=P(2)/2
140 LET S=PEEK 16400 + 256*PEEK 16401     - address of Z
150 DIM D$(32)                    - string for significand
160 INPUT A$                      - main loop in program
170 LET Z=VAL A$
180 GOSUB 300
190 GOTO 160
300 FOR I=2 TO 5                  - calculate binary value as .s and 1s
310 LET T=PEEK (S+I)              - byte of significand
320 FOR J=1 TO 8
330 LET D$(8*I-J-7)="."
340 IF INT(T/2)*2 <> T THEN
                 LET D$(8*I-J-7)="1"
350 LET T=INT(T/2)
360 NEXT J
370 NEXT I
380 PRINT A$; "="; Z; "EXP="; PEEK (S+1)    - last value is exponent
390 PRINT D$;
400 RETURN
```

The stored exponent for P(I) is I and its value is 2**(I-129) for all the permitted values of I (1..255). With this program, one can see that dividing 2**(-128) by 2 gives 2**(-128) and in general a value with an exponent which underflows by just one is set equal to P(1)=2**(-128).

The values given by the routine MACHAR give a characterization of a

floating point unit. This characterization has the advantage of being easily determined. An alternative characterization of floating point is that given by Brown [10]. The Brown model is excellent for deducing the properties of a system but it is very hard to ensure that the paramoter values of the model are correct. In fact, the obvious parameter values for the ZX81 do not work because of a defect in the normalization in the add operation. This defect is easily demonstrated by use of another one, namely that 0.5 is converted to one bit less than the true value. On the other hand, dividing 1 by 2 gives the exact result, so

```
PRINT 1/2-0.5, 0.5-1/2
```

gives

```
2.3283064E-10     0
```

In the Brown model, this defect causes a single bit penalty in parameters which then become:

Radix=2
Significand digits = 31 (=32-1 for penalty)
Exponent maximum = 126
Exponent minimum = -128

Note that this characterization does not cater for rounding. In consequence, it is rather pessimistic in this case. The above parameters have not been checked which ideally should be done with [7], but (as noted above) the work involved in putting this 12,000 line FORTRAN package on the ZX81 would be prohibitive.

The normalization defect does not occur with the BBC micro. Hence the characterization of the BBC machine is

Radix = 2
Significand length = 32
Exponent maximum = 126
Exponent minimum = -128

The arithmetic operations

Since Standard Basic does not have an integer data type, it is important that the infix operations should give integer results for integer-valued operands when the result is in range. The Pascal test suite contains a program to test floating point by use of integer operands (less than the significand length). Such a test is not appropriate to Basic. Simple tests do reveal correct results for +, -, * and ABS when the results are in range. The Pascal test suite has a special test for division since this operation is sometimes approximate for "unusual" hardware. For instance, on the Cray 1, 15/3 does not give 5 exactly because the operation of division is performed as multiplication by the reciprocal. Since 1/3 is not representable, the exact answer is not given. This technique would not be acceptable for Basic; the two microprocessors give the correct result for divisions of this type.

A significant problem arises with the exponentiation operation **. This operation is not present in Pascal and in consequence the Cody and Waite test for the FORTRAN operation is not present in the Pascal Validation Suite. In fact, X**Y is implemented as EXP(Y*LN(X)) on the ZX81 which is unsatisfactory for the following reasons. Firstly, X cannot be negative which is not appropriate if X is a negative integer. Secondly, integer-valued operands do not always give integer-valued results when the latter are in range (for example, 3**2 is not computed as 9). Thirdly, as Cody and Waite point out (page 84), the accuracy is not good for results of large or small magnitude. In defence of the ZX81, it must be noted that an accurate version for ** is a large and cumbersome routine (and hence the space it would consume could be more useful for other purposes). The BBC version of ** was not tested.

Square root

The square root function is called SQR in Basic, whereas sqr in Pascal is the square function. The test program of Cody and Waite can easily be translated into Basic by inserting the constants from MACHAR. In fact, the main loop reduces to 21 lines of code. The results are

2000 Random values in range 0.5 to 1.0

	ZX81	BBC
Maximum loss in binary places	= 1.60	0.99
RMS loss in binary places	=-0.02	0.0

2000 Random values in range 1.0 to 2.0

Maximum loss in binary places	= 1.61	0.99
RMS loss in binary places	= 0.14	0.0

The negative value for the RMS loss on the first test is caused by an inadequacy in the algorithm used combined with a very small loss in accuracy. In this case, results are rounded so that the result is accurate to less than one place. Such figures are replaced by zeros in the subsequent tables. All these results pass the criteria used in the Pascal validation suite. The special value tests were also handled successfully.

Logarithm

The natural logarithm function is the same in Pascal and Basic so the Pascal test could be used directly. Four different tests are involved of 2000 random values. On the first attempt to run the test of 2000 random values very near 1.0, large apparent errors were detected. This was a false indication due to the coding to subtract 1.0 from the argument. This coding was written in Pascal (and FORTRAN) as

LET Y = (X - 0.5) - 0.5.

The purpose of the double subtraction is to avoid the loss of the bottom bit. Unfortunately, 0.5 is not represented exactly on the ZX81 due to a defect in the read routine (see Page 6), so a large apparent error was indicated. Replacing the 0.5 by 1/2 solved the problem. It is not clear if this change should be made to the Pascal tests, since independent tests are performed for reading constants (including 0.5). However, an optimizing compiler could upset this statement, so the logic inserted in the Pascal version of MACHAR should be added here (i.e. call of the function "st"; now added to version 3.1 of the Pascal validation suite).

The test results are summarised as follows

	ZX81	BBC
2000 values very nearly to 1.0		
Maximum loss in binary places	= 1.86	19.48
RMS loss in binary places	= 0.10	14.13
2000 values in range 1/square root(2) .. 15/16		
Maximum loss in binary places	= 2.12	4.81
RMS loss in binary places	= 0.54	2.84
2000 values in range square root (0.1) .. 9/10		
Maximum loss in binary places	= 2.74	4.32
RMS loss in binary places	= 0.77	1.92
2000 values in range 16..240		
Maximum loss in binary places	= 0.99	0.99
RMS loss in binary places	= 0.00	0.00

These results pass the criterion used in the Pascal validation suite, except for the first test on the BBC machine. The special value tests were also satisfactory for this machine.

Sine and Cosine

The sine and cosine functions are the same in FORTRAN, Pascal and Basic in that they have an argument in radians. The three tests of Cody and Waite are for SIN in the range 0..PI/2, SIN in the range 6*PI..(6+1/2)*PI and COS in the range 6*PI..7*PI. These three ranges are those for the usual 2000 random values to which are added special argument tests.

The special argument tests and the first set of random values passed the usual criteria. However, of the other two tests, two failed quite badly on the ZX81 and one on the BBC. All these cases of failure were due to incorrect range reduction, as could be seen by investigating the values which gave the least accurate results. The range reduction is part of the logic of these functions to reduce the argument value to 0 .. PI/2. The error in the range reduction was confirmed in the ZX81 as follows. Since the value used for PI is stored in the monitor, the range reduction algorithm can be determined. In fact the range reduction of X is performed by the equivalent of

```
LET X = X - 2*PI*INT(X/(2*PI))
```

The actual value used in the above formula may be PI or PI/2, but this would make no difference with the radix of 2. The equivalence of the range reduction to the above was shown by confirming that SIN(X) is computationally the same as SIN(X - 2*PI*INT(X/(2*PI))).

The second SIN test has been repeated on the ZX81 with the improved method of range reduction recommended by Cody and Waite. For this test, the arguments must be reduced by subtracting 2*PI and 6*PI which is done as follows.

In the second test where the range reduction involves subtracting 2*PI from the argument value one performs:

```
LET X = X - 3217/512 + 2*C,
```

and in the last test where one must subtract 6*PI one performs.

```
LET X = X - 9651/512 + 6*C.
```

This approach effectively gives 9 extra bits of precision since the rational value is computed exactly. The value C is 0.000008909089793.

The results obtained were as follows:

	ZX81	BBC
2000 values for SIN in range 0..PI/2		
Maximum loss in binary places	= 1.97	2.00
RMS loss in binary places	= 0.27	0.24
2000 values for SIN in range 6*PI..6*PI + PI/2		
Maximum loss in binary places	= 16.82	4.75
RMS loss in binary places	= 11.34	0.57
2000 values for COS in range 6*PI..7*PI		
Maximum loss in binary places	= 15.81	11.91
RMS loss in binary places	= 10.41	6.68

The fact that the loss is a consistent one for the last two tests can be illustrated by plotting the loss against the argument value. The loss is large when the true value is near zero.

Exponential

The coding of the FORTRAN version of the Cody and Waite test of the exponential function EXP in Basic is straightforward. Three tests of 2000 random values are performed. The first set of values lies in the primary range giving a result between 15 and 20. The second set lies in a large negative argument range and the last in a large positive range. On the ZX81, only the first test passes the criteria placed upon the results included in the Pascal version. All the tests passed on the BBC machine.

Since the usual method of calculating exp involves splitting off the exponent of the floating point representation, and reducing the range, it would appear that the range reduction is suspect. Cody and Waite point out the difficulty in carrying out the range reduction as it is likely to lead to large relative errors for arguments of large absolute magnitude.

A program was written for the ZX81 to display graphically the relative error as a function of the argument value. The sine argument reduction errors showed a consistent trend but the errors for exp showed no such consistency. In fact, the cause of the error was the range reduction since the results were markedly improved by inserting the followiing range reduction algorithm recommended by Cody and Waite.

```
400 REM GOSUB 400 CALCULATES E = EXP(X)
410 LET N = INT( X/LN(2) + 1/2 )
420 LET E = EXP( X - N*355/512 + N*2.129444E-4 )
430 LET Q = 2
440 IF N < 0 THEN LET Q = 1/2
450 IF N = 0 THEN RETURN
460 FOR K = 1 TO ABS(N)
470 LET E = E * Q
480 NEXT K
490 RETURN
```

Lines 450-480 could be abbreviated to LET E=E*Q**N if ** were itself calculated by repeated multiplication.

The second and third tests were much improved, see the summary table. The basis of this range reduction is the same as that for SIN/COS, i.e. a better approximation to the range size by means of an exact rational plus a much smaller term.

The results obtained were as follows:

	ZX81	BBC
2000 values for EXP in range -0.284 .. 0.346		
Maximum loss in binary places	= 1.96	1.00
RMS loss in binary places	= 0.17	0.00
2000 values for EXP in range -65.1 .. -3.46		
Maximum loss in binary places	= 5.78	2.67
RMS loss in binary places	= 4.85	1.31

2000 values for EXP in range 6.93..69.3

Maximum loss in binary places	= 5.78	2.77
RMS loss in binary places	= 4.85	1.59

Arctangent

This function is called arctan in Pascal but ATN in Basic (although it is marked as ARCTAN on the key board of the ZX81). The Cody and Waite tests can be coded in Basic without difficulty. Four sets of 2000 range argument values are tested. All these tests pass on both machines according to the criteria used in the Pascal validation suite.

The results obtained were as follows:

	ZX81	BBC
2000 values for ATN in range -0.0625..0.0625		
Maximum loss in binary places	= 1.85	2.18
RMS loss in binary places	= 0.20	0.27
2000 values for ATN in range 0.0625..0.268		
Maximum loss in binary places	= 2.04	1.97
RMS loss in binary places	= 0.31	0.52
2000 values for ATN in range 0.268..0.414		
Maximum loss in binary places	= 1.91	2.30
RMS loss in binary places	= 0.41	0.96
2000 values for ATN in range 0.414..1.0		
Maximum loss in binary places	= 1.89	2.63
RMS loss in binary places	= 0.23	0.71

Read

Cody and Waite do not consider the reading of decimal constants (in data or programs). Their omission has been partly filled in the Pascal validation suite as follows. The exact decimal values of I*2**(-30) for I in the range 1..1000 are written to a text file. These values are then read by the Pascal read procedure, using the same routines as before for assessing the accuracy. The STR$ and VAL functions of Basic allow this long Pascal test to be coded much more simply. For the 1000 values, the maximum relative loss was 2.52 binary places and the root mean square loss was 1.09 places on the ZX81. The exact value was obtained every time on the BBC machine.

The problem with the logarithm test showed that the read function was hardly satisfactory and yet the results obtained above passed the criteria used in the Pascal validation suite. The success with the tests of read is surprising since they have failed on two main-frames. The reason for the results lies in the actual algorithm used and also in the properties of the floating point system. The algorithm can be determined for the ZX81 by writing a similar routine in Basic until a perfect match in terms of rounding errors is obtained. The following program demonstrates the logic which works by a simple left to right accumulation of the value.

```
 10 FOR J=1 TO 10
 20 LET A$=STR$ (100*RND)
 30 GOSUB 200
 35 LET Y=VAL A$
 40 PRINT X; " "; Y; " "; X-Y
 50 NEXT J
 60 STOP
200 REM CONVERT A$ TO X
210 LET X=0
220 LET Y=1
230 FOR I=1 TO LEN A$
240 IF A$(I)="." THEN GOTO 310
245 IF A$(I)="E" THEN STOP
250 IF Y<1 THEN GOTO 280
260 LET X=10*X+(CODE A$(I)-CODE("0"))
270 GOTO 320
280 LET X=X+Y*(CODE (A$(I))-CODE("0"))
290 LET Y*Y/10
300 GOTO 320
310 LET Y=1/10
320 NEXT I
330 RETURN
```

The exponent is handled by multiplying the significand by an integral power of 10 (or dividing for a negative power). Note that this algorithm gives the correct result for integers up to the size of the binary significand, which is essential for Basic.

The rounding errors are incurred on the addition and multiplication on line 280 and the division on line 290. The rounding error is bounded because after 10 digits the product on line 280 will no longer contribute to the sum. However, the forward summing is poor because all the rounding errors on lines 280 and 290 will contribute to the value for the first 10 significant digits. The rounding ensures that the result is not biased and the radix of 2 minimises the error. This algorithm on a hexadecimal machine with truncation (such as the 370) would be disastrous.

Conversion of the ZX81 program for testing reading to the BBC machine

proved calamitous. The minimal Basic standard does not define enough string handling routines to program this test conveniently. On the ZX81, arrays of characters and strings are identical allowing a simple program for this test. On the BBC machine, the two are different - indeed one can have A$ as an array of characters and another A$ in the same program as a string! One can index only an array of characters, and the function to convert a string to a numeric value works only on a string. In consequence, the test was coded completely differently on the BBC machine. It is not clear that any method would work correctly on both systems.

Write

As with read, Cody and Waite do not consider the conversion of floating point values to decimal constants. The Pascal validation suite uses the same approach as for read. The computed values I*2**(-30) for I in the range 1..1000 are output to a file using write. The output is then read as characters and the resulting value computed with the true result by subtracting the values in character form. By this means, the error in the output conversion can be found without rounding error. Again, the STR$ and VAL functions in Basic make the use of files unnecessary. In Pascal, the write output can request 30 decimal digits so that perfect results are possible. This is not possible with ZX81 (and BBC?) Basic; indeed, the format is always floating point for the range of values under test. Also, all decimal output is rounded to 8 significant figures (9 on BBC), which is less than the approximately 10 places of the floating point representation. The effect of this choice is that computations appear to give 8/9 figures (the output format). Judged in this light, the mathematical functions appear to be very good. For instance, executing PRINT SQR(N*N) for small integers N always results in N being printed. Some calculators adopt this philosophy in having guard digits that are not displayed.

On the ZX81 machine, the results for the 1000 values gave a maximum relative error loss of 7.55 binary places and a root mean square loss of 5.36 binary places. Needless to mention, these results fail the criteria adopted in the Pascal validation suite. The routine is probably unbiased: 566 values were too small and 434 too large. (A chi-squared test fails at the 0.01 level but the values used are not independent.) The largest error was just a small amount in excess of a truly rounded

result. Hence the rounding process itself has an inconsequential error. The additional decimal place on the BBC machine gives results which only just fail: 4.39 places for the maximum relative error and 2.77 for the rms error.

The actual algorithm used for write is unknown. Algorithms are available which would produce perfect results in this test, but they are expensive in computer time. Interestingly, write (or rather STR$) is one of the slowest functions on the ZX81.

Copy

The Pascal validation suite contains an additional test of both read and write. Here 100 values of the usual form are repeatedly copied and the result of the 50th copy compared with the computed value. If a read or write routine is biased, then such a test could show a drift, as has been observed on one system. With the rounding that is performed on STR$ in Basic, thi~ test is uninteresting since it gives essentially the same results as write. All the losses are on the first write and from that point onwards, read gives the exact converse of write.

The random number generator

The random number generator is not strictly one of the functions tested, but it was used heavily in the tests. The quality of the random number generator is not critical. In fact, the generator is very simple on the ZX81 and is given on Page 34 of the manual. Written in Basic, one can check it as follows:

```
10 LET B = RND*65536+1
20 FOR I = 1 TO 20
30 LET B = B*75
40 LET B = B - INT(D/65537)*65537
50 LET X = (B-1)/65536
60 IF X <> RND THEN PRINT "NE"
70 NEXT I
```

No output was produced. Hence the values have only the top half of the significand set, the rest being zero. Fortunately, the tests perform

calculations on the generated numbers to calculate the argument value so that this defect did not invalidate the tests. Of course, the generator used in the Pascal validation suite could have been used [9]. The BBC random number generator sets more than the top half of the significand and has no obvious defects.

Summary

The errors observed with the tests can be summarised as follows, where MRE is the maximum relative error loss in binary places and RMS is the corresponding root mean square loss. The values in brackets are those obtained by the author for the ZX81 using the improved range reduction or by Dr A C Norman by modification of the BBC monitor. A failure according to the Pascal validation suite criteria is denoted by F. This failure occurs when MRE>4 or RMS>2.

	ZX81		BBC	
Function	MRE	RMS	MRE	RMS
SQR	1.60	0.0	0.99	0.0
			(0.50	0.0)
	1.61	0.14	0.99	0.0
			(0.99	0.0)
ATN	1.85	0.20	2.18	0.27
			(0.25	0.0)
	2.04	0.31	1.97	0.52
			(0.99	0.0)
	1.91	0.41	2.30	0.96
			(2.51	0.55)
	1.89	0.23	2.63	0.71
			(0.99	0.0)
EXP	1.96	0.17	1.00	0.0
			(0.99	0.0)
	5.78F	4.85F	2.67	1.31
	(2.84	0.27	2.67	1.48)
	5.78F	4.85F	2.77	1.59
	(2.01	0.18	2.88	1.73)
SIN/COS	1.97	0.27	2.00	0.24
			(0.0	0.0)
	16.82F	11.34F	4.75F	0.57
	(2.11	0.33	0.0	0.0)
	15.81F	10.41F	11.91F	6.68F
			(1.92	0.03)
LN	1.86	0.10	19.48F	14.13F
			(0.25	0.0)
	2.12	0.54	4.81F	2.84F
			(0.99	0.0)
	2.74	0.77	4.32F	1.92
			(2.51	0.55)

	0.99	0.0	0.99	0.0
			(1.0	0.0)
READ	2.52	1.09	0.0	0.0
WRITE	7.55F	5.36F	4.39F	2.77F
			(1.26	0.0)
COPY	7.55F	5.41F	4.39F	2.77F
			(1.17	0.0)

The totally different pattern of errors shows that the routines are quite different in spite of using identical representations and both *coming from the Cambridge area. The revised BBC figures will be those* obtained from systems to be released shortly.

Acknowledgements

The author would like to thank Dr A C Norman for the additional BBC results and Dr M G Cox for carefully reading a draft of this report.

References

[1] Cody, W J and Waite M. 'Software Manual for the elementary functions'. Prentice-Hall, 1980.

[2] BS6192: 'Specification for computer programming language Pascal' British Standards Institution. 1982.

[3] American National Standard Programming Language FORTRAN, ANSI X3.9-1978.

[4] American National Standard for minimal Basic, ANSI X3.60 1978.

[5] Wichmann, B A and Ciechanowicz, Z J. 'Developing the testing procedures', in Pascal Compiler Validation, Wiley 1983.

[6] Logan, I, 'Understanding your ZX81 ROM', Melbourne House, 1981.

[7] Schryer, N L. 'A test of a computer's floating point unit', Computer Science Technical Report No 89. Bell Laboratories. 1981.

[8] IEEE. 'A proposed standard for binary floating point arithmetic',

ACM Signum Newsletter. October 1979.

[9] Wichmann, B A and Hill, I D. 'An efficient and portable pseudo-random number generator', Applied Statistics Algorithm 183. Vol 31, No 2, 1982.

[10] Brown, W S. 'A simple but realistic model of floating point computation'. Computer Science Technical Report No. 83. Bell Laboratories. 1980.

DATE
ILME